

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

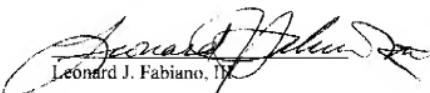
In re Application of: Fabiano III, Leonard Serial No. 09674221 Filed: 02/13/2002 For: "Electronic Commerce Systems and Processes, Especially for the Cable Television Industry"	Group Art Unit: 3622 Examiner: RETTA. Yehdega Customer Number: 25854 Confirmation Number: 3271
---	---

AFFIDAVIT PURSUANT TO 37 C.F.R. 1.131

1. My name is Leonard J. Fabiano, III. I am over twenty-one years old and I make the following declaration based on my own personal knowledge.
2. I am the inventor of the invention disclosed and claimed in the above-referenced patent application, which is currently assigned to EMediaTrade, Inc.
3. I conceived and reduced to practice the above-referenced invention by no later than November 22, 1999. This is evidenced by the documents attached hereto as Exhibits A, B and C.
4. The document shown in Exhibit A was printed on April 21, 1999. The document shown in Exhibit B was printed on November 3, 1999. The document shown in Exhibit C was printed on August 5, 1998. By no later than November 22, 1999, I delivered these documents to James L. Ewing, IV, the attorney who filed the above-referenced patent application.
5. I invented every aspect of the invention as claimed in the above-referenced patent application by no later than November 22, 1999.
6. Each responsible person was diligent in pursuing patent protection for this invention at each stage of the disclosure approval process and the patent application preparation and filing process.

I declare under penalty of perjury that the foregoing is true and correct.

5/13/2003
Date


Leonard J. Fabiano, III

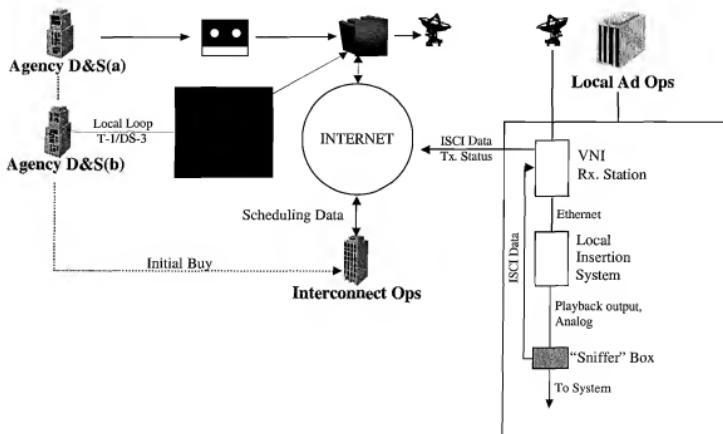


INTERCONNECT PROCESS REVIEW

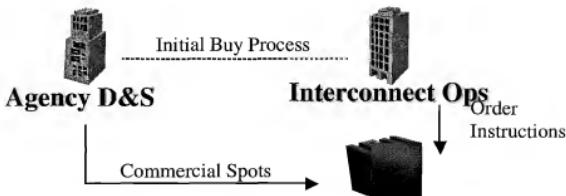
The interconnect solution, as proposed by VNI, provides hard interconnect functionality to a metro market without the expense and limitations imposed by traditionally implemented hard interconnects. As a bonus, it is far more affordable and can be implemented more quickly than a terrestrial fiber solution.

Functionality includes the ability to distribute schedules and spots, gather verification information, handling of agency orders electronically, and simplified spot reconciliation against orders to only list a few. The purpose of this document is to walk through each of these steps and break them out into stages. Throughout, assumptions, special features and requirements will be specifically addressed. To conclude we'll address interconnect software functionality in a project VNI calls IIMS (Integrated Interconnect Management System), that inter-operates with VNI's existing ECTracker and MediaTracker platforms.

ARCHITECTURE OVERVIEW



* This diagram shows media being submitted to VNI in two ways. The first (a) is the more traditional method of air courier service. Once at VNI the content is encoded and the Spot ID is assigned, according to its ISCI code. The other method (b) shows video submitted to VNI using our frame-relay network. With this method Agencies have fixed bit-rate encoders (provided by VNI) that automatically encode and submit video to VNI. Once at VNI the spot is transcoded into the proper format and the Spot ID is added.

STEP 1: GETTING CONTENT TO VNI

Assumptions:

- Total guaranteed interconnect inventory availability within each local market is a maximum fixed percentage.

Special Features:

1. Since a percentage of a local systems inventory will be available for interconnect ad placement, negotiating for space is not required. It's already available and can therefor be placed into the market more quickly.
2. MarketTarget product will need to be developed by VNI in order to provide advertisers with valuable, detailed data about the markets available for purchase.

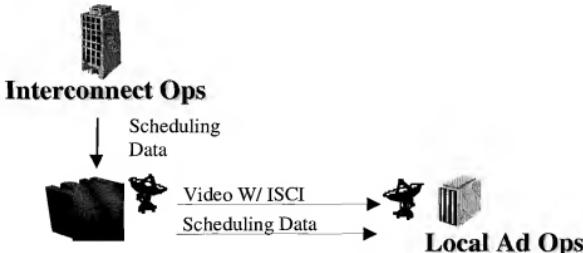
Explanation:

- Video from the agency (D&S) and order instructions from INTERCONNECT, are submitted to VNI's Network Operations Center (NOC). Since there are standards discrepancies with today's local MPEG-based insertion systems, VNI supports different platforms for export such as MPEG2 Program Streams, MPEG2 Transport Streams and MPEG 1.5. Maintaining customer local site profiles at VNI to ensure the correct MPEG file is delivered in the correct local system format is central to enabling this feature. This process is functioning and available today. The ISCI code will also be embedded in the data stream for later use in verification. Ads will remain in inventory until a buy is associated with the ad (ISCI code).

Once in the proper format, and working in conjunction with IIMS, a media delivery scheduler will build multicast groups. VNI will create these groups based upon daily INTERCONNECT buys and deliver the spots over satellite. Delivery verification and status information will be available to INTERCONNECT real-time throughout the delivery process. If for any reason video cannot be delivered within a defined set of parameters (i.e. three tries within 3-hours) VNI will send the spot on tape to that site. (*VNI make goods for misses?*)

The regions identified for placement will have been chosen based upon a hybrid ratings tool developed by VNI that compiles market demographic data and ratings

information to facilitate optimized market buys. This pre-buy tool can be available on-line by accessing a sub-link within MedaTracker. In turn this data can be used by agencies in conjunction with existing schedule building tools from the interconnect. **This tool does not exist and requires development.** In the future, this tool should be enhanced to provide even more accurate market data by increasing the market sample size. This type of project will require the participation of Nielson, SMART or AdCom so that at a minimum the top markets will have very accurate market response data.

STEP 2: GETTING SPOTS / SCHEDULE INFORMATION TO LOCAL MARKETS

Assumptions:

- All regional/local receive points have broadcast quality advertising insertion equipment and respective traffic and billing functionality.

Special Features:

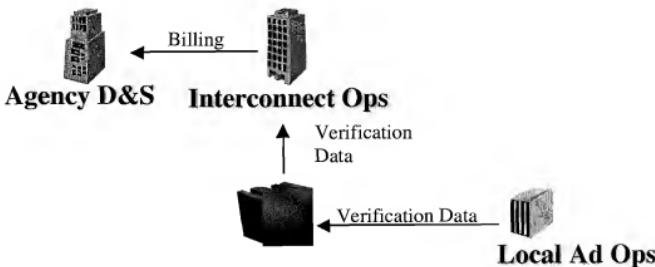
- Orders into these markets from the NOC occur instantaneously.
- Since INTERCONNECT inventory is guaranteed and a large number of systems are available to order into, the process of Road Blocking within a market becomes simple.
- There will be a 24-hour buy window for INTERCONNECT ad placement. If an avail held for a interconnect spot has not been fulfilled within 24-hours of going to air, the local operator reserves the right to place its own local content in that slot.
- Local site hardware provided by VNI
 - .9m satellite dish (similar to Direct TV)
 - VNI Receive Station
 - DVB receive card
 - Pentium processor based
 - MPEG2 decoder(optional for local playback)
 - 56k Modem (internet access)
 - Storage
 - Ethernet card
 - Upload interface (for SeaChange systems)
 - Data router

Explanation:

- Sites will receive their media via satellite. Once received at the ASO or even subtending headend, a VNI utility notifies system personnel that spots have been delivered. The spots will be imported into the MVL or local video archive and

matched against the scheduling data received electronically prior to video transmission. If additional subtending sites are present and connected via fiber, customers will conduct business as usual from this point. Otherwise they may wish to have VNI deliver the spots directly to the subtending sites. If desired, the process of delivering content to subtending sites automatically based upon orders to specific ASO's / regions can be done by VNI.

If at the local level the spot is renamed, this will not effect the verification process as the ISCI code is imbedded in the playback stream. VNI suggests implementing this process to eliminate problems associated with ASO's or local headends changing the spot title or ID in the T&B system.

STEP 3: THE VERIFICATION PROCESS

Assumptions:

- There are 916 headends within the Top50 markets alone. Some of these will be connected to ASO's through pre-existing network communications.

Special Features:

- This part of the process requires the use of what VNI terms a "Sniffer" box. This third party verification tool will be incorporated into the solution and maintained by VNI.

Explanation:

3. A "Sniffer" box will be placed on the back-end of the insertion playback device and continually searches for embedded ISCI codes in the playback stream. When a code is detected, it will record the time, ISCI, length of playback and network. This data will be saved and the logs sent to the ASO, then on to VNI. These uploads can be scheduled to happen at regular intervals throughout the week (i.e. once daily at 7:00 PM) and can be done over the Internet using the provided ISP account that is part of all VNI satellite receive station configurations.

Once the ISCI verification data is back at the NOC, VNI will reconcile the data against the delivery instructions to immediately determine if make-goods are necessary (VNI currently supports such functionality through its Affidavit Editor). This data is also immediately routed to the interconnect for processing where the data will go through another reconciliation against the agency order, then processed for billing.

INTEGRATED INTERCONNECT MANAGEMENT SYSTEM

T&B-type software functionality is required at the NOC location. This software (IIMS), which currently does not exist in whole, plays a key role in handling information throughout the previous defined processes. The functionality is primarily spread between

VNI and interconnect with functionality required at the local level as well. Work needs to be completed in integrating the different stewardship systems into these operations.

Below is a list of the seven key components of the IIMS software. Keep in mind that a number of the listed functions may already exist in one form or another, in varying degrees, in VNI's existing MediaTracker and ECTracker products.

1. Inventory Management: IIMS will manage a database of video and corresponding traffic data for every piece of interconnect spot video placed onto the INTERCONNECT interconnect. In placing ads, IIMS will ensure that INTERCONNECT management is only aware of its reserved inventory and that all orders are cleared against the same. From the local level, summarized INTERCONNECT usage can be made available through a simple web interface.
2. Electronic Orders: Incorporating some of VNI's existing product solutions, IIMS will be capable of receiving and placing electronic orders from Agencies and to interconnect in a manner requiring no manual re-entry. All orders are processed and cleared against INTERCONNECT inventory. This order information is then forwarded to the local systems. Electronic summary confirmations throughout the process can be provided by VNI where required.
3. Schedule Optimization: This feature is a predictive tool that analyses the impact of a buy order, then recommends means to accommodate the buy while providing a snapshot of the buys effects on existing and remaining buys. Using variables such as ordered dayparts, networks, and geography, spots can be shifted in the schedule to allow others to clear unless prevented by buy parameters. Used by INTERCONNECT, if this optimizer cannot clear an order as requested, an electronic change request may be forwarded to the buyer with suggestions of what can be bought that will clear. This way it becomes much easier to manage buys across the interconnect while enabling optimal usage.
4. Schedule Dissemination (VNI has already completed significant portions of this functionality): This requires advanced capabilities at the NOC, interconnect, and at the local operators. Each day the local operators download orders and modifications for INTERCONNECT using the VNI network. Since all transactions are electronic, re-keying of INTERCONNECT contracts is not required, and data is automatically transferred into the local system. INTERCONNECT contracts are input into the local systems T&B software at the highest priority thus automatically preempting local spots (assuming it's within the 24-hour buy window). From a simplicity standpoint, INTERCONNECT contracts contain no reference to the advertiser or agency and thus local operators do not have to establish or maintain a T&B database. Furthermore, contracts will not contain price information so that the local affiliate is not tempted to preempt INTERCONNECT spots due to apparently higher rate local advertisers.

5. Copy Instructions: Copy instructions are downloaded each day using the VNI network, and processed with the incoming INTERCONNECT contracts and modifications. These copy instructions, received electronically utilize the agency-assigned ISCI codes that uniquely identify each commercial. Physical copy and their pertaining instructions are under INTERCONNECT's control.
6. Verification/Makegoods: Using data gathered by the local "sniffer" box, the IIMS system will reconcile the information against the orders placed. This can be done in a number of ways but fundamentally it will first gather (or the local system will submit) the information from the local sites and compile it regionally or by ASO. Next the regional data will be pulled into VNI, processed, then immediately sent to the interconnect for final reconciliation and billing purposes. These log files will be sent to the interconnect on a daily basis. If in the process of gathering this data the system finds a spot did not run for whatever reason, the IIMS system will re-schedule makegoods following pre-determined fulfillment measures. Included in this is notification to the sales associates so they can issue makegood requests to the buyer. With verification taking place daily, in-flight makegoods are very possible yielding much higher fulfillment than typical interconnect spot buys in cable.
7. Now that the interconnect has its summary verification information, the final reconciliation process will begin using the interconnect's order records. Invoices may be created in summary or detailed form as required by the agency. Invoices are returned electronically to the agency via the VNI network.

SOFTWARE INTERFACE SAMPLES

The following interface samples were pulled from VNI's existing ECTracker and MediaTracker products.

TTV & VNI Discussion

11/3/99

Vision

A world leader in providing targeted network and e-commerce application services for the distribution and management of digital content.

Company Overview

- Founded in November 1996
- Blue chip investors
 - AT&T Ventures
 - BankAmerica Capital Company
 - Alliance Technology Ventures (ATV)
 - U.S. Ventures Partners (USVP)
 - Institutional Venture Partners (IVP)
 - Kinetic/Arete Ventures
 - UPS
- Based in Roswell, GA

TTV Objectives

- To begin understanding and developing sales of a national scale targeted medium in cable
- To learn about future node and subscriber level targeting through developing sales, research, and verification tools for this project
- To change the agency community's perception of cable to one of being "out-in-front" with new technology
- To develop a fulfillment platform for national cable which enables it to hit \$1B/yearly revenues on-schedule
- To provide a platform for rapid launch of market-wide interconnects

Options for Achieving Objectives

- **Hardware Solution**
 - Purchase insertion hardware for each headend
 - Data network to distribute spots to each headend
 - Coordinate inventory usage with local T&B Systems
 - Establish central operations facility for T&B
- **Software Solution**
 - Configure existing digital inserters to accept dual-schedule operations
 - Data network to distribute spots to each headend or ad-sales office
 - Coordinate inventory usage with local T&B systems
 - Establish central operations facility for T&B

Hardware Solution Analysis

- **Strengths**
 - Faster initial deployment
 - Less dependent on local operations personnel
 - Beginning infrastructure in place for MSCO centralized and national applications
- **Weaknesses**
 - Both capital and maintenance costs are drastically higher
 - Software solutions required regardless of hardware expense
 - Hardware depreciable life is short and new technology is moving rapidly

Software Solution

- **Strengths**
 - Drastically lower costs upfront and ongoing
 - TTV & Industry's software and back office capabilities improve
 - System is scalable from 2% of inventory to handling all national spot traffic
 - Plays off the strengths of each party, Local operations and National sales
 - Being prototyped in Hartford market interconnect
- **Weaknesses**
 - Longer time to market
 - Software development always has some risks
 - Agency perception that "less has changed" about the cable industry in a software solution

Skyconnect

- **Strengths**
 - Good insertion equipment
 - Has developed some good market alliances
 - System is more “open” than Seachange’s
 - N3 money as backing
- **Weaknesses**
 - Weaker software development capability
 - No in-house satellite networking experience
 - No ability to traffic, track, schedule or distribute TTV buys
 - Insufficient development resources to support their rapid growth
 - Weak customer service

Searchange

- **Strengths**
 - Good insertion hardware
 - Good insertion application software
 - Large market share of existing equipment
 - Deep understanding of ad-insertion technology industry issues
 - Stability of a “small-to-midsize” public company
- **Weaknesses**
 - Permanently closed system architecture
 - Inability/unwillingness to work out interoperability & partner with other vendors
 - Unsuccessful T&B software experience
 - No in-house satellite networking applications
 - Poor customer service

StarNet/CAMS/Radius

- **Strengths**
- Experience operating Philadelphia Interconnect
- Have a pretty good T&B package
- Have some good software development talent in S.F.
- **Weaknesses**
- In-house technology development always on the edge of “abandonment”
- High price
- No satellite data-networking experience remaining in-house
- No insertion hardware experience remaining in-house
- Spotty track record on commitment to industry projects

VNI

- **Strengths**
 - Open Systems
 - Creating **open** systems where none existed before
 - Deep knowledge of industry and TTV Issues
 - Significant in-house software and data networking experience
 - New leadership, credibility and significant funding
 - Large project experience
 - Applicable patented technology
- **Weaknesses**
 - Not a T&B vendor
 - Not an insertion equipment vendor
 - Past leadership issues tainted market perception
 - New company or traditional start-up company risks

Standard VSAT Solution

- **Strengths**
- “Always-on” network
- Less “hassle” managing local phone lines & ISP accounts at sites
- Slightly lower monthly return-data expense vs. Internet
- **Weaknesses**
- VSAT hardware is an extra \$1.5M capital with a somewhat slow return
- Max outbound data rate 2 Mbps
- Very narrow “shared” return-path bandwidth (1 Mbps = 700 bps per site)
- Field service of VSAT gear
- Routing options expensive

VNI Internet Return Solution

- **Strengths**
- Much lower upfront hardware costs
- 56Kbps dial-up return path vs 700bps shared VSAT channel
- System is a routed data network
- Phone lines and ISP connections are managed by VNI personnel
- Integrated with VNI Middleware and Network Management systems
- **Weaknesses**
- Not an “Always-on” network
- Return path monthly expense slightly higher than VSAT

VNI VSAT Return Solution

- **Strengths**
 - Always on network
 - * Higher return data throughput
 - * Higher outbound data rate
 - * Data routing included in price
 - *Flexible hub location
 - Newer technology than traditional VSAT
 - On-site maintenance provided
- **Weaknesses**
 - Up front equipment costs higher than Internet model
 - Return data slower than Internet model
- * = Advantages of VNI VSAT solution vs traditional VSAT

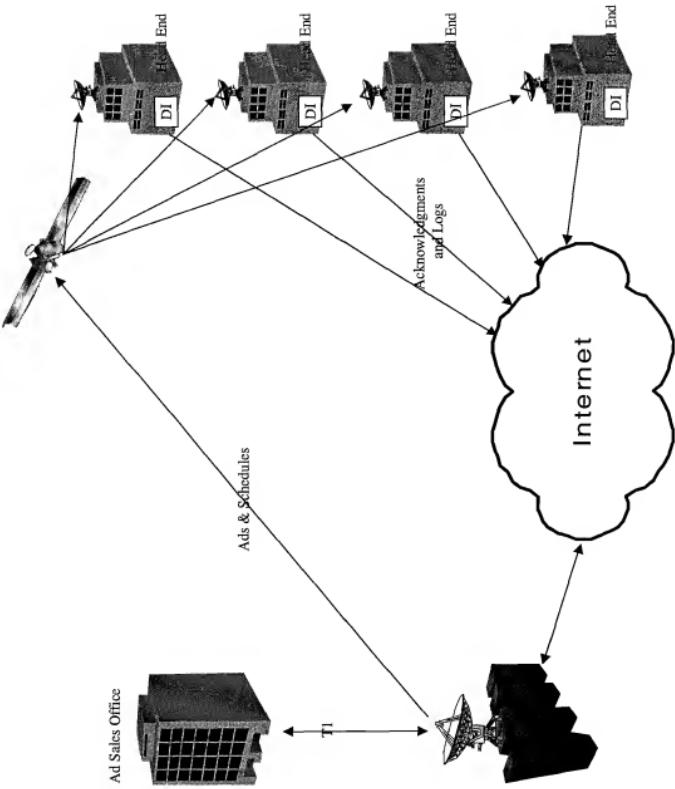
TTV – Interconnect Approach

- Current National Spot revenue over \$500M with 80% of national spot revenue from top 50 markets
- Thus national spot value in the Top 50 markets is near \$400M annually
- NCC Interconnect CAGR above 66% for 3 years
- Therefore, with 34 markets (treating them equally) unconnected, the opportunity cost is \$180M each year. Even with industry growth rate averages factored in the opportunity cost is \$125M per year.

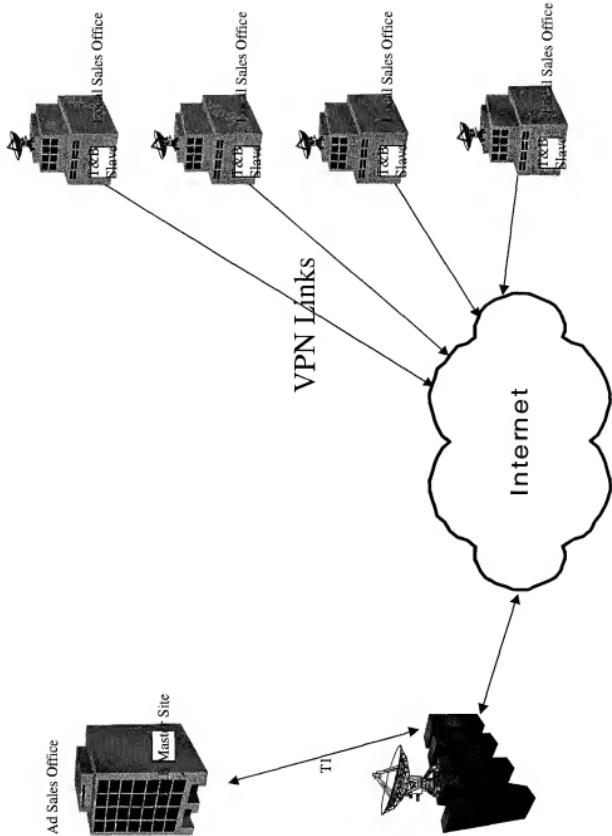
New TTV Approach

- Build out interconnects in the 34 markets NCC has identified ASAP
- Use existing technology to move quickly, meaning Insertion hardware, satellite data, and existing software systems
- Develop TTV software and applications over time and interconnect the Interconnects...

Regional Hub/Interconnect Video

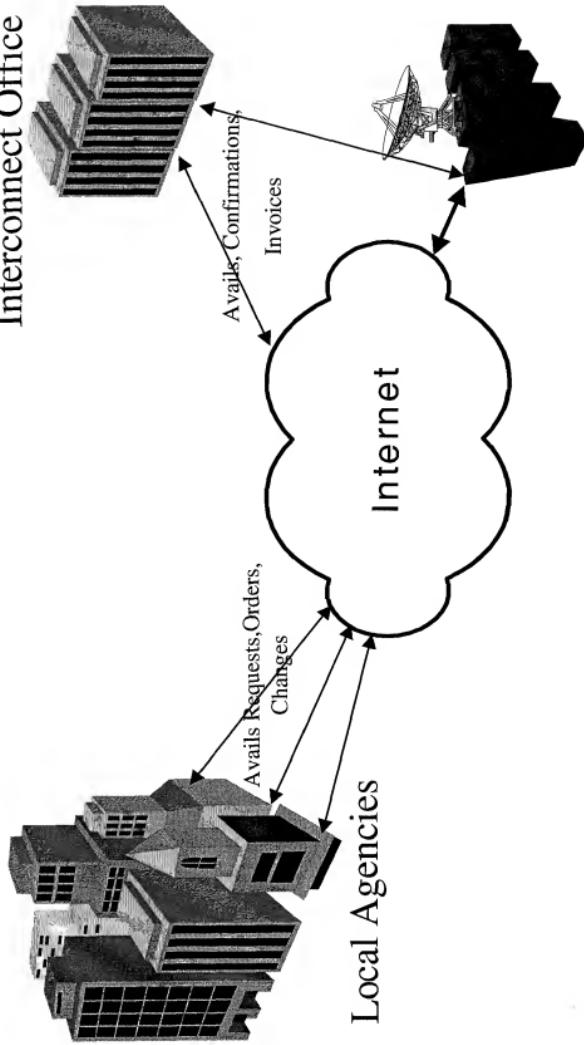


Regional Hub/Interconnect T&B



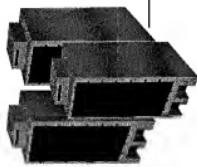
Regional Hub/Interconnect

E.T.M.

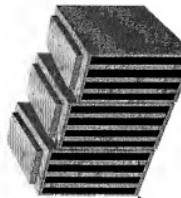


MSO/National Spot Submission

Agencies or Production Houses



Ad Sales Office



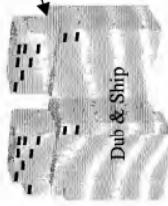
Counter Tapes

T1

VNI
Frame or Internet

T1

DSS
Service



Dub & Ship

Tapes

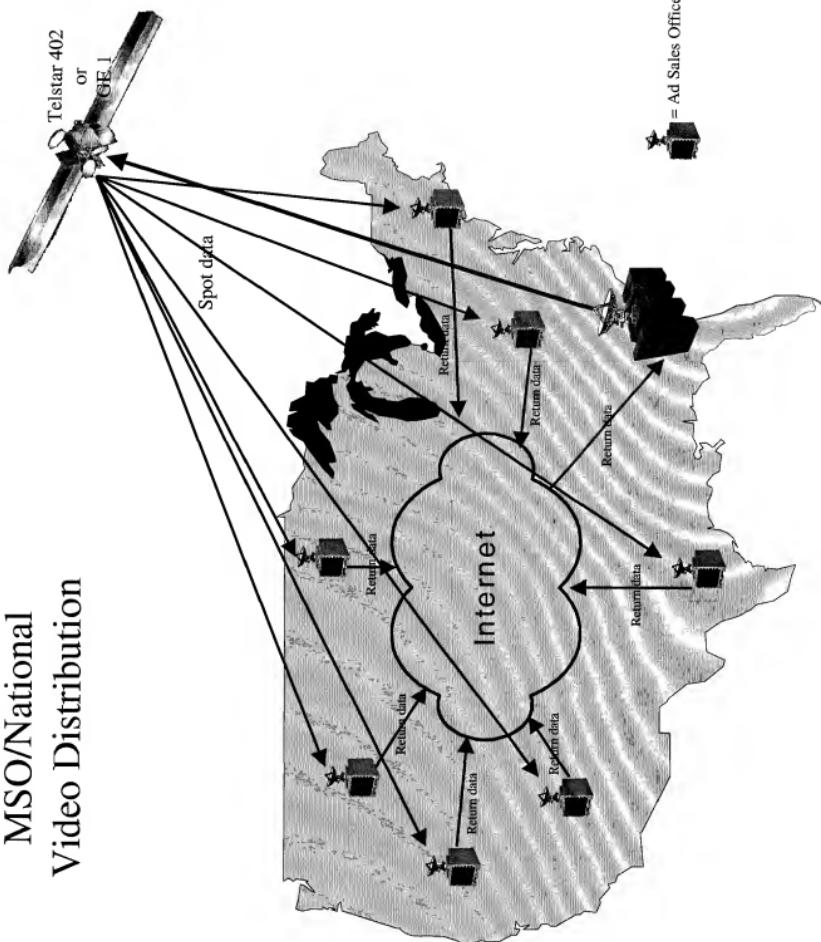
Overnight
Tapes



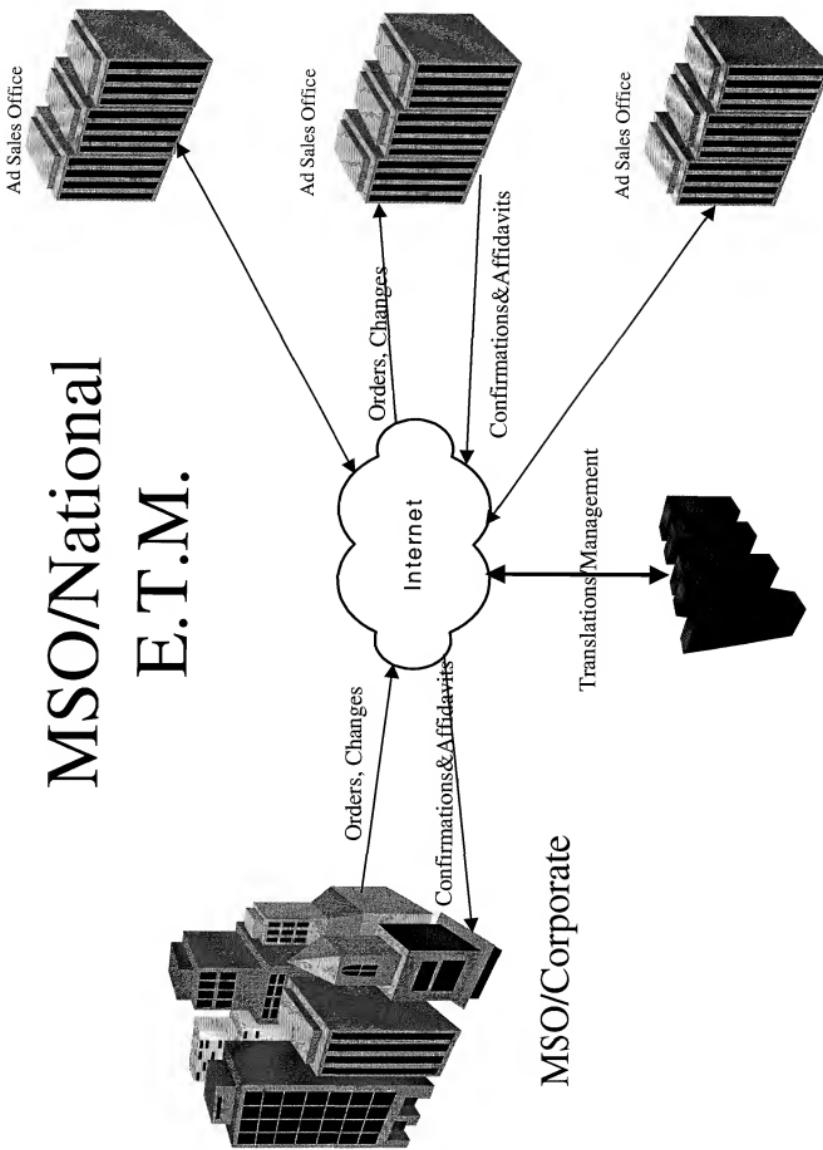
Overnight
Tapes



MSO/National Video Distribution



MSO/National E.T.M.



Video Networks, Inc.

**8601 Dunwoody Place
Atlanta, Ga. 30350
(770) 998-8872**

Electronic Commerce

The Application

August 5, 1998



Overview

Our Mission

- Build an application that immediately serves the business document communications needs of the trading partners in the national spot cable market.
- Perform the above task in such a thorough and competent manner as to grow the revenues of our local cable partners so that they're wholly convinced of the value of our service.
- Gain a decisive market share to make competitive entry in this narrow field relatively unattractive.
- Leverage our foundation application, our knowledge, and our relationship to the cable and agency partners in national spot cable trade into use for the local media trade.

Influential Forces

Considerations of our business mission drive the architecture of our automated solution. Some of the key considerations include:

- We intend that our electronic commerce customers should require little or no adaptation of their current business processes, automation systems, and trading partner relationships in order to use our services. It is our assertion that facilitating direct document exchange between applications is the best way to accomplish this feat. Our customers will concern themselves with translations or mappings of any sort. We take documents from and deliver them to our customers in formats that are native to their own automation systems. In cases where there is little or no automation, we will provide lightweight applications. These will allow potential users of our service to create, edit, and manage electronic documents with very little investment of time or capital.
- If we do not require our customers to change their trading relationships in any way in order to use our service, they will more likely use it.
- When our services are completely non-intrusive to our clientele requiring little or no system maintenance, it won't require any support staff on their part, thus further reducing their expense and difficulties.
- The fact that using our service only requires the capability to run browser software will maximize its availability by minimizing entry requirements.
- We expect our services to our customers to be intuitive and simple to use, much easier than their existing trading schemes.
- We would like prospective customers to be able to enroll easily over the Internet, without requiring any special installation or training to immediately engage our services to transport simple documents that they key directly.
- We expect our system to respond rapidly to customer requests or input. (Barring bandwidth issues for Internet based customers)
- We expect our system to perform reasonably well for light clients over the Internet.
- Both our private and Internet connection customers will enjoy secure document trading.
- We expect to extend support for new document types and trading partners with relative ease.
- As documents and document procedures change, we expect that these changes to be made quickly without inconveniencing our customers and at minimal cost to ourselves.

Well...at least my client is thin!

When we say the word "client", in the "client/server" context, we refer to a software application, rather than a customer (In our specific case our client software happens to be used by our customers, or clients, but this is irrelevant to our discussion.). In this context, the client software is a client, or satellite, of the server software. Our client software will allow users to create, edit, and perform all kinds of functions relating to electronic documents, but the documents will reside on our server(s).

Given the maintenance and support concerns over the far-flung nature of our EC enterprise, coupled with our desire to make our application as non-intrusive as possible to our customers, we need to keep our client applications as lightweight, or as "thin", as possible. To maintain software in a constantly functional state on literally hundreds of computers simultaneously would be a daunting task using conventional methods.

We plan to build this infrastructure adhering to a strict division of labor between the client and server. The client software will primarily be concerned with document presentation, capture, and controlling management operations (not performing management operations). This implies that much of the functional weight of our system will be borne centrally by servers, which will reside in Atlanta.

The only requisite support software, for our client applications, will be a Java capable web browser. All of our client programs, that the user will see, will be uploaded to their web browser on demand. To reduce network traffic the server will only reload applications, that have been previously uploaded, if they are changed. When we change a client program, we will load it on the server and then anyone who tries to use it will receive the update automatically.

Although the users will edit and invoke document operations through their client software, the documents themselves will reside on our server(s), where the most significant operations against them will be performed. When a user views, prints, or edits a document, it will be uploaded at that time to their web browser. Commands issued to save or transmit a document are executed at the server. In other words, the client applications permit a user to view or control operations on a document but the real work takes place on the servers.

Where do documents come from Mommy?

Of course users of our client software will be able to create new documents, like contracts or orders, on-screen, should they so desire. However, we expect and hope to be a better solution than this for **Error! Bookmark not defined**. users who have in-house automation systems. These in-house automation systems, such as Traffic and Billing and Agency Stewardship programs, lie at the heart of our clients' businesses and are perfectly capable of producing many thousands of paper documents. These are exactly the kind of documents our customers will send and receive electronically.

As stated previously, in a not-so-electronic document that we produced, it is core to the very premise of **Error! Bookmark not defined**, to move documents between these automation systems, delivering them in native forms. This will avoid the re-keying time, clerical errors, transport delays, and other issues that challenge our customers' current document exchange methods. So wherever possible, we will send and receive electronic documents directly to and from to these automation systems without requiring any format or value translations. This begs the question, "How do we move documents from one customer's automation system to another customer's automation system if all we have in between are two client applications that can't do anything but display documents?" Probably the best way to answer this question is through a real world example.

Out with affidavits

Harry, a billing operator at a local cable provider, has just finished his reconciliation processes at the end of the broadcast month. Before he does anything else, he wishes to get the national spot bills out to his rep firm. Harry is about to enter that mysterious realm of electronic commerce:

- ❑ To begin with, Harry generates electronic affidavits for his national spot trade with his T&B system. It puts all of them for the current period into one big file we'll call an affidavit batch.
- ❑ Through his **Error! Bookmark not defined**. client software, Harry logs into the VNI server.
- ❑ He chooses the option to upload electronic affidavits.
- ❑ Using Win95's "File Open" dialog, he points to the file which contains the entire batch of electronic affidavits. He presses the "Upload" button. The batch is sent via a file transfer operation directly to the EC@VNI Server.
- ❑ Once the batch is received intact at VNI, a transaction is initiated on the server that parses the batch of affidavits, breaking it down into individual affidavit documents.

- Each affidavit is translated into the **Error! Bookmark not defined.** core affidavit format and assigned a unique document ID, which will identify it, throughout its life, both to Harry and to the system.
- The affidavits are moved into Harry's outbound mailbox, where they are held until he takes further action.
- Based on the data that the server has stored about Harry's electronic trading partners, it is able to assign destinations to all of the affidavits from the batch except one.
- Harry can now view any or all of the documents he just transmitted with his client software. Although it doesn't matter so much to Harry where the documents physically reside, whenever he requests to view a specific affidavit, by selecting it from an on-screen list, it is transmitted to the client application, which then displays it for him. (Because edits are turned off for affidavit documents, he is prevented from making any changes them.)
- Note that the Harry is viewing the affidavits while they reside on our server in **Error! Bookmark not defined.** core affidavit form, as opposed to a form that is unique to him or any other party. This allows us to use a single viewing and editing program for everyone wishing to view affidavit documents.
- In reviewing the documents, Harry realizes that the affidavit for Charlie's Shoes has an error. So he goes back to his T&B system and re-creates an the affidavit for Charlie's Shoes and once again uploads an affidavit batch, except this time the batch contains but a single affidavit for Charlie's Shoes.
- The server is smart enough to know that it can't have duplicate invoices from a single provider so it automatically replaces the erroneous invoice with they newly corrected one.
- Harry is now satisfied with all of the affidavits except one that's missing a target address. Harry realizes it's a new client that the system has never seen before so he selects the target himself from a list of candidate targets.
- With all of them appropriately addressed, Harry highlights all of the outbound affidavits in his viewer and issues a single "Transmit" command for the entire batch.
- The **Error! Bookmark not defined.** server generates a new affidavit, with a unique document ID, for each affidavit sent by Harry to each target.
- The new affidavits, which are copies of the ones sent by Harry, are moved into the target rep firm's inbound directory, but remain in the **Error! Bookmark not defined.** core format.
- The server performs a value mapping operation on these new affidavits, which replaces Harry's unique values with the rep firm's unique values. In one case, Harry calls Headline News Network "HDLN" and the rep firm calls it "HLN". The server performs this mapping so that the affidavits look natural when viewed by someone from the rep firm.
- The server moves the original affidavits to Harry's "sent" mailbox and changes their state to "Sent".
- With the sending process completed, Harry is finished, for the time being.

Now It's the rep's turn

- It just so happens that, Susan, the billing manager at the rep firm decides to check their **Error! Bookmark not defined.** in-box. A large number of affidavits have arrived from Harry.
- After a quick review, Susan decides they look good, as far as she can tell!
- So she highlights all of the affidavits in her in-box and issues the "Download" command.
- The server dutifully copies and translates each affidavit into the format expected by the rep firm's automation system and packs them into a single file (one big batch), and downloads them immediately.
- The affidavit documents are moved on the server from the rep firm's "In" mailbox to their "Received" mailbox.
- Meanwhile, the rep firm's automation system sees all of these new affidavits, in its favorite flavor, and happily consumes them until it sees Charlie's Shoes, where it does a little gagging.

- Susan's automation system tells her it likes all of the affidavits excepting one. She returns to her **Error! Bookmark not defined.** client and acknowledges all but the single affidavit for Charlie's Shoes..
- The **Error! Bookmark not defined.** server receives the acknowledge command for these affidavits and goes back and marks each of their originating affidavits from the sender (Harry), as "Acknowledged" and forwards an acknowledgment to him.
- In the meantime, the billing manager gleefully issues the "Reject" command against the wretched affidavit from Charlie's Shoes.
- Error! Bookmark not defined.** promptly responds by creating a "Rejection" document, in the **Error! Bookmark not defined.** core format of course, assigns it a unique ID and places it in Harry's in-box.
- When Harry opens his in-box the next time, he sees that his whole affidavit batch was acknowledged except for Charlie's Shoes, there in place of a merry acceptance, he sees the bold letters of REJECTION!
- Not to worry! He quickly corrects yet another billing error in his T&B system and regenerates the affidavit.
- In his **Error! Bookmark not defined.** client he hurriedly chooses to the "Upload" command, pointing out exactly which document he's replacing.
- The system as before uploads it, translates it to **Error! Bookmark not defined.** core format, and stores it with the same document ID as before. Since it was already targeted to his rep-firm, he rapid fires by once again issuing the "Send" command.
- And the process repeats as before, except this time Susan only uploads a lone affidavit for Charlie's Shoes. Once her automation system has digested the Charlie's Shoes' affidavit without heartburn, she begrudgingly issues the "Acknowledge" command.
- Harry sees that the Charlie's Shoes' affidavit has finally been acknowledged. He takes a deep breath and slowly releases it. He knows that tonight he'll get his first real sleep since the start of billing week.

Prominent Features

Document Centric

One of the underlying themes of this system is the fact that it centers on documents, as opposed to X12 transmission batches or application file batches. Our thought is that users deal better with documents than they do with batches. Group and Interchange IDs mean very little to someone who is trying send affidavits to their rep firm. They just want them to arrive.

When there's a question, as to the payment status of a particular invoice, noone really knows what interchange, group, batch or any other ID it was assigned (other than perhaps the invoice number). However, they do know that it was the January affidavit for Charlie's Shoes. Our approach in giving our customers document management tools, in addition to a communications infrastructure, is somewhat unique (and not just in the media world). It will be a lot like the overnight delivery services that can track your package's delivery. But our system will allow the users to check on it themselves immediately, without having to know any codes.

"E-Mail Like" User Interface, but more secure

The EC@VNI Client software, in many ways, will look like an e-mail system. It will have "In" and "Out" boxes just like an ordinary e-mail system. Users will also be able to review any documents they receive on-screen before and after sending or receiving them. They will also be able to edit some pieces of information within certain documents when they're in a non-sensitive state. In order to protect the audit trails of interactive information systems, users will not be prevented from making potentially compromising edits to certain documents.

Group Operations

To make things easier for our users, any operations that can be performed on a single document may be performed on groups of documents. So if Harry selects 27 affidavits from his "Out" box, he can press the "Send" button once and all 27 affidavits will be agency (or rep) bound.

Custom Document Views

Outside of the normal "In", "Out", "Sent" and other state-based boxes, that are integral to each mailbox, each **Error! Bookmark not defined.** user can define their own document views. As an example, a billing manager at a rep firm may wish to create a custom document view that shows affidavits from a specific trading partner. A system administrator for an agency may wish to view all of the documents received from a certain trading partner in the last quarter. Once a user defines a document view, it remains in the system as long as they wish.

Account Management and Administration

Another unique feature of our approach is that a trading partner may have many assigned accounts. They may assign each of their users to a different account or have dedicated accounts to a specific purpose, such as receiving affidavits. In this case multiple authorized users might have access to that affidavit account. This will also allow us to deliver orders to a specific buyer within an agency and still retain the concept that the trading partner is the agency itself, and not a single buyer within the agency.

Each trading partner will have at least one user that is assigned as their administrator, who will be authorized to perform administration tasks:

- Assigning which document types may be sent and received to and from each account.
- Determine which users have access to which accounts.
- Assign which document types a user may access within an account.
- Maintain trading partner lists and cross-references.
- Update value map cross-references.

In addition, an administrator may create document views that can include documents from any and all accounts, while a normal user will only be able to create views on documents from the mailboxes to which they are assigned.

Comments

An authorized user may attach any number of public or private comment lines to any document. Public comments are transmitted with the document for viewing by the target users, while private comments are not.

Document History

All documents will remain on file, although they won't be in the active "Inbound" or "Outbound" state directories once they've been transmitted. They may remain on the server for up to 90 days after they have been completely resolved. Some documents, such as affidavits, may remain on file pending remittance advice indefinitely. The 90-day clock will only begin ticking once remittance advice has been received.

Customer Oriented Translations

In a typical EDI implementation, each consumer of the electronic data is responsible for translating from the format in which it is received to a format they're able use. On the outbound side (when they're sending documents) they're responsible for reformatting the documents into a standard format that nobody's information system really understands, but is an accepted transmission standard that all of the translation systems can speak (with a lot of help). All of these translations can be costly in terms of manpower, training translation systems, and other areas.

Our system will automatically reformat documents when they're downloaded, to a format native to the target's information system. When a user of our system sends documents, they will upload them to us, in their application's native format, and our system will automatically translate from that. This way our customers won't have to be involved in any translation or deal with its complexities. Our customers will receive documents and import them directly into their information systems and export documents directly from their information systems.

Intelligent Value Mapping

In moving documents between trading partners' applications, there is much translation that has to occur, not just in the format of documents, but in their content as well. In normal EDI implementations management these content translations, or Value Mappings as we call them, are automatically and intelligently performed by our system. As an example, Harry calls The Cartoon Network, "TOON". Susan calls it "TCN". If Harry and Susan were exchanging electronic documents, using normal EDI methodologies, each would have to know what the other calls it and each would have to perform a value mapping on incoming documents. In our system, we know what each calls The Cartoon Network and we automatically translate between them. So even though Harry sends an affidavit that says "TOON", when Susan receives it, it will say "CTN".

Data Synthesis, Filling in the blanks

Few software application systems carry all of the key data elements that may be required for another trading partners' systems. Therefore, in traditional EDI, the people who are creating the translation maps have to fill in a number of holes, and in some cases they have to fill in the holes manually. In the relatively short life of our company we have seen several instances where data required by agency systems doesn't exist in the station's application systems. This data must be either be captured or synthesized before sending it on to the target.

We have found it to be the case that these missing data elements have been supplied somewhere in the document exchange cycle. However the data elements aren't always captured by the trading partner's system because there was either no convenient place to put it, or somebody forgot to key it and their system doesn't tell them they forgot.

For example, let's say Harry, a trusty operator at a station, receives a contract, from a rep firm, for a national advertiser. The rep firm has included the agency's estimate number on their printed contract.

When entering the contract into their T&B system, Harry has no place to input the agency estimate number, so he leaves it off. Three months later, when Harry sends a paper affidavit (through the snail mail system) back to the rep firm, it's missing the estimate number.

Susan is really miffed, being the billing manager at the rep firm, because she has to lookup the agency's estimate number, from the contract that was sent three months ago, just so she can send a paper invoice to an agency that includes their estimate number.

Using the EC@VNI system, everyone will be happy because when we forward the original contract to Harry, we'll skim off the estimate number and save it. When Harry confirms, to the rep firm, that he's accepted the contract, we'll skim off his contract number and tie it to the agency's estimate number. Three months later, when Harry sends an affidavit (electronically through EC@VNI), our system will automatically place the estimate number on Harry's affidavit so Susan won't have to make that emergency trip to the drug store for more Valiums.

Intelligent Addressing

Just like an e-mail system, some documents need to go to more than one place. EC@VNI will support as many as 256 targets for a single document. When the user initiates the "Send" operation it will send a copy specifically suited and value mapped for each target.

Even more impressive is our ability to know who the targets of each document when they're uploaded to us. When a user uploads a batch of documents from their information system to EC@VNI, in much the same way that it performs value mappings and data synthesis, **Error! Bookmark not defined.**, will address the documents appropriately. EC@VNI will store tables of each sender's trading partners along with cross references to their keys for those trading partners, and automatically assign targets to each of the documents. The user always has the option of changing or removing a target, and when they do so, it will update EC@VNI's cross references.

For example, Susan uploads a whole batch of contracts destined to various stations around the country to EC@VNI. The EC@VNI system knows that Susan's information system calls Joe's Cable in Peanut, Georgia - station 17546. So when EC@VNI sees a contract from Susan destined to Station 17546, it immediately assigns Joe's Cable as a target. Susan can still add another target, or direct it to a different one altogether, before she sends it.

Automatic Document Validation

Usually, in EDI, when a document's target is pretty picky about the way they receive their data, the sender has to be very cautious about the way that they format and translate their documents before they send them. Thus many EDI shops have developed complex procedures for sending documents to select trading partners. When we know this is the case, **Error! Bookmark not defined.**, can automate document validation that will notify the sender when they attempt to send documents that violate the target's criterion. The sender will even be able to perform a preliminary validation on their documents prior to sending them. In this case the system will immediately notify them whether they meet their targets' qualifications.

Transmission Acknowledgment

In normal EDI trade, parties must exchange Functional Acknowledgment documents to confirm that the other party received it. Each receiving party is responsible to notify the sender that they indeed received it. Since our system manages the whole exchange, it will automatically flag the sender's document when the receiver acknowledges it. The receiver acknowledges or rejects a document with a simple mouse click or keystroke. In this case the sender gets an almost tactile feedback, they don't have to sit and wonder if and when they'll receive an acknowledgment.

The states of their documents are immediately updated as soon as the receiver presses the "Acknowledge" button. By visually scanning the contents of their "Sent" box (state directory) they'll know which documents have and have not been acknowledged.

Automatic Document Generation

In a normal exchange of documents between media trading partners, most people have to work through their primary information system, which create documents that can be exchanged. In some cases this is time consuming and inconvenient.

Let's use the example of Harry receiving a contract from a rep firm. He must first key the contract into his T&B system, and then print a paper confirmation, and then mail it back to the rep firm. Let's say Harry signs up for EC@VNI so he no longer has to manually key his contracts from the rep firm. But his information system isn't capable of producing electronic confirmation documents. So, in this case, he's still stuck with his printing and mailing routine.

Harry suddenly remembers that EC@VNI will auto-generate a confirmation for him from the contract he received. So instead of printing and mailing, he brings up his "Received" box, on the EC@VNI Client, and highlights the contract from the rep firm. After pressing the "Gen Confirmation" button, he is prompted for his contract and advertiser numbers. As soon as he enters them and presses "Send", an electronic confirmation is on its way back to the rep firm.

In this example, a confirmation is sent in response to a contract without touching an in-house information system directly.

Transaction Integrity

All data processing systems experience some amount of down time. In a robust data processing system, downtime is tolerated through redundant fail-safe systems. Such is the case with EC@VNI. In addition to hardware based redundant systems, the software will be built with the concept of a transaction rollback. Should the system fail mid-stream in carrying out an operation on a document, on re-initialization it will return all documents, and their attendant data, back to their state prior to system failure. Operations can then be re-invoked without any residual effects.

Security

All documents moving to and from the EC@VNI system will be protected through the latest in secure systems technology. All data is encrypted as it is sent and received.

The EC@VNI Server system is completely secure not allowing any unauthorized access.

In summary, we're way better than EDI

It is clear from the previous example that **Error! Bookmark not defined.**, will be a robust business communications and document management system that literally will extend the capabilities of our customers. The days of printing, stuffing envelopes, postage meters, snail mail, and redundant data entry are nearing a close in the media world. Our solutions will so change the way our customers operate that five years hence they will look back and wonder how they ever managed without it.

The **Error! Bookmark not defined.** system, and the service it facilitates, goes way beyond traditional EDI implementations. Our users will enjoy more benefits than those organizations supporting the expense their own EDI shop. They will also enjoy these benefits without enduring any of the difficulties. All of this is possible because EC@VNI is looking at the big picture of media partner communications and the information systems that exist today. We're not waiting for in-house information systems or proposed document standards to evolve at all. Our users can have their legacy systems connected today without waiting (So long as today doesn't come before 2nd quarter of 98).

You may wish to stop here

Unless you're the technical type, you probably won't want to read too much further into this document. The remaining portion consists of technical discussions as to how the **Error! Bookmark not defined.** system will be built.

Platforms

It will be Java for our clients

The client software will be entirely conceived in Java applets. The driving forces behind this decision are:

- Instantaneous compatibility to virtually every client operating system is one of Java's more attractive features. We won't have to force any of our customers to change computing platforms (except those few using TRS 80's).
- Java's ability to run with applets uploaded to a web browser eliminating the need to distribute the client application using media such as diskettes, CD-ROMs, etc.
- A rich graphical interface support will allow us to create applets that run from a web browser but aren't limited to the scant functionality of an HTML type language. They can literally look, feel, and act like typical Windows or Mac applications, with windows overlaying windows rendering more 3D, rather than 2D, performance.
- Java readily and simply supports data communication over networks.
- Being an object-oriented language, Java permits us to easily create reusable classes for other applications.
- It is a widely supported, rather than a proprietary language which won't leave us dependent on single vendor and where we'll enjoy the benefits of a language advanced by mass market appeal. There are also numerous Java programmers to be found with those ranks growing daily.

Serve us some UNIX

The server operating system of choice will be UNIX due to the following:

- We already have much of our electronic commerce applications running under the UNIX operating system.
- UNIX is supported on some of the more powerful computing platforms in the world. It offers us the scalability, fault tolerance, and throughput that will be required of our industrial-strength application.
- Our Sybase SQL license is for a UNIX platform.
- We have much in-house expertise in UNIX.
- UNIX has built-in and robust support for all of the TCP/IP based communications that we'll be using.
- Multi-tasking is something UNIX was designed to do from its inception.
- Out of the box UNIX sports massive amounts of text processing utilities and scripting languages that will prove very useful in our electronic commerce applications.
- The Sun systems we own only run UNIX.
- The world's greatest text editor, "Vi", comes standard with every UNIX system.
- Bottom Line...UNIX kicks ass!

Real programmers use C++

The core modules in the **Error! Bookmark not defined.** server system, such as the Document Manager and Value Mapper objects, will be rendered using C++:

- Given that we have the potential to be connected to hundreds, if not thousands, of clients, the core objects in the **Error! Bookmark not defined.** server application will require as much throughput as we can muster. C++, when skillfully written compiles to very fast native executing code.
- C++ is very supported on all UNIX platforms.
- There are an abundance of C++ programmers.
- We have much experience in-house using C++.

- C++ connects very well using the UNIX communications facilities.
- You can readily hook up to Sybase and other SQL databases using very standard ODBC drivers or even embedded SQL. It's probably better to use ODBC but in cases where speed counts we can fall back to embedded SQL should the situation warrant it.
- C++ has excellent support for reading and writing with streams on local devices.
- C++ is a very object oriented language, which allows us to create readily reusable classes.

SQL like a pig?

The database tables for the server application will reside on a Sybase SQL system because:

- We already use Sybase SQL for our other applications and own a copy.
- Sybase is supported by powerful UNIX systems giving us our much-needed performance.
- It readily supports ODBC and drivers for it are widely available.
- Sybase has an excellent reputation for support and have remained a front-runner in the highly competitive SQL database market for many years.

Windows on billing

Although this decision hasn't been finalized, it appears that our billing system will be running on a Windows NT server. Since we are not developing the accounting applications in-house, we are constrained to the platform best supported by our accounting software supplier. Given our needs, the most attractive accounting software systems run on NT. The interface from the **Error! Bookmark not defined.** will likely only take place on a periodic basis, at best nightly, and will not require extended high bandwidth.

Additionally, EC operations department will not manage the accounting system, like they will the **Error! Bookmark not defined.** Server applications. So the most attractive choice may be to have the accounting system running on a completely separate server, operating independently of EC. This would greatly limit access to the accounting software and database and possibly render the more reliable and secure

Don't worry, it's written in the script

We already own a significant amount of electronic commerce software. It will be used everyday for translations to and from X12 formatted documents and for other translations that don't require the throughput of a native executable. The **Error! Bookmark not defined.** Server application will use UNIX scripting languages to tie these existing EC applications, our native executing objects, and other text processing utilities together into cohesive, cooperative transactions.

Client/Server Communications

For the foreseeable future, communications between **Error! Bookmark not defined.** Client and **Error! Bookmark not defined.** Server, will take place over TCP socket connections. Should it prove necessary in the future to implement load balancing and other advanced features, it may be necessary to use something that layers over TCP Sockets, such as CORBA.

In summary, it's a mixed bag, but way cool!

It's true that we're using several languages and, counting our client application, a number of operating systems to deploy this system. However the reasoning behind each choice is solid and was chosen after much investigation and research.

The only software that our end-users will see runs under a single language and always in the context of a web browser. The back-end, consisting of the **Error! Bookmark not defined.** Server, will be completely invisible to the user. To them it will look like a very uniform system that runs on their desktop. Although Java is a very clear choice for the desktop it's not our language of choice for the server.

Error! Bookmark not defined. Server

A skeleton mostly

In review of the Overview and Features sections of this document, one could easily gain the impression that **Error! Bookmark not defined.** server is going to be a substantial application. This is true, but what is curious about the design is that the core of the system is not going to provide all of the functionality heretofore expressed.

To illustrate, the list of things that has to be done to move a document between trading partners can be short or long, depending upon the requirements specific to each information system (or lack thereof). In some cases almost nothing has to be done, excepting moving the document from the sender to the target(s), while in others, we may have to perform extensive validation, complex translations, large batch collations, etc. It is therefore difficult to create a far-reaching and uniform structure that will efficiently handle all of these different document management and exchange scenarios. So to jump to the heart of the matter, we won't!

Specifically, the **Error! Bookmark not defined.** server system will be a skeleton application that provides a number of core services, like communications and document tracking, but the validations, translations, collations, and even document routing will come from smaller individual applications that, in essence, put the meat on the skeleton. **Error! Bookmark not defined.** server provides the glue to tie these sundry applications together into a cohesive framework.

Storage Strategies

All things happen at the server

All documents will be stored persistently on the server. No documents will reside at the client except temporarily for editing and viewing purposes. The server will perform all operations. Even when a client is editing a document, it will only be saved, as far as **Error! Bookmark not defined.** is concerned, at the server.

Use a Database Manager only when necessary

One of the overwhelming concerns in building the server system is throughput. Given the potential of many hundreds of users and perhaps hundreds of thousands of documents, the server must perform all operations as efficiently as possible. On the other hand, database managers (DBMS) are wonderful tools that can allow us to perform ad-hoc queries against all kinds of data, but in utilizing them you pay a high price in performance, setup, maintenance, and accessibility.

Where data needs to be stored persistently, we will not store and organize it in the DBMS unless it meets one following criterion:

- The primary data object is of a global nature, in that it can't be organized systematically within a single trading partner, mailbox, and/or user. An example is a document, a document ID needs to be assigned to each document globally so that we can refer uniquely to each. However, the entire contents of a document do not really need to reside in a database. Another example, a user needs to be defined globally because they will each have their own login ID, password, etc. Both of these data objects share a global context where as a document view, is only used within the context of a single user. We can therefore organize document views under the context of a single user and have persistent file-based storage designated exclusively per user.
- The data will likely be needed for ad-hoc queries where the queries have the potential of global context.
- A relationship to other data objects needs to be enforced that can not be easily enforced without using an onerous and proprietary scheme.

VNI Form Documents

All documents, excepting transmission batches, will be stored on the server in a VNI standard format. Each document type that moves through **Error! Bookmark not defined.** Server will have its own VNI standard format. When an incoming transmission batch is received at the server, it will immediately broken down into individual documents and stored in their VNI form. **Each VNI form document will occupy a single file, each VNI file will hold but a single document.** Only document transmission batches, that have been uploaded to the server or are to be downloaded from the server, may contain more than a single document per file. Whenever an **Error! Bookmark not defined.** Client requests a document, we will transmit it in its VNI form. We will always translate to and from VNI form documents.

The VNI form of any document type will contain all of the data elements required by all trading partners for the same type. So the VNI form of a document type will contain a superset of the data elements required by any single trading partner. The VNI form of an affidavit, for example, will contain all of the data elements required by DDS, CCMS, Compulink, Datatech, etc. Therefore, from a VNI affidavit we should be able to derive an affidavit satisfactory to any of these automation systems.

VNI form documents will also be mappable. This is to say that they will be organized into records; identified by the Record ID residing in first data element of each row and delimited by the newline ('\n') character; and data elements (fields), variable length delimited by the pipe symbol '|' within each row (record). So we will be able to refer to any specific record by its ID and any data element by its Record ID and position, i.e.; Record XYZ, Position 6.

VNI format documents will contain only ASCII characters, no non-printable characters, no binary data whatsoever. All floating point numbers, integers, BCD, etc. will all be converted to strings.

Several benefits will result from this approach:

- The **Error! Bookmark not defined.** client will only have to contend with a single format for any document type. This will cut way back on the necessary development for the client software.
- It reduces the number of translation engines that we have to create and maintain. To illustrate, let's say we have affidavits coming from 3 different automation systems, A, B, and C. We have 3 different target automation systems for outgoing affidavits, X, Y and Z. If we didn't have a core document format, we would have to maintain translations for A to X, A to Y, A to Z, B to X, B to Y, etc. We would have 9 translations. Let's add another affidavit source and destination for a total of 4 in each direction. We now have 16 separate translations to maintain. If one trading partner changed their format slightly, we would have to update now 4 translators for one minor change! Using a core format with 4 senders and 4 targets, we have 4 incoming translations and 4 outgoing translations for a total of 8. If one target changes their format slightly, we update but a single translator.
- Given that the VNI form will be mappable, we can create generic document utilities that will function in a table driven fashion (See *Document Definition Files* below) for any trading partner. For example, we can build a value mapper that will replace the sender's values with the target's based on simple coordinates. So to change the network names for a document, we could tell a program that it has to swap networks values around, it could know that for affidavits, the network values are found in record 51, position 10. It can then in a network cross-reference table see which values each partner uses and quickly swap them. If we stored affidavits in 4 distinct formats, we would have to have 4 separate programs just to swap network values.
- Documents will be editable with any text editor or and even viewable with a simple screen dump.
- We can refer to and programmatically operate any specific data element within a document without incurring the overhead of storing them in a database. We can still retrieve them quickly from the disk using lightweight streaming mechanisms. We may still store select key elements in a database so that we have the benefit of executing queries against the key elements.
- We will be able to manipulate the documents with simple shell script programs.
- Documents in this form will translate more readily to X12, EDIFAQ, and other standardized EDI formats using our off-the-shelf translation software.

Standard Document Record Types

One or more of the following record types may appear in a VNI form document. These record types are consistent across all types of documents. Each provides information that may be used system wide.

- Document Header (00)** – Each VNI form document stored on the system will begin with a document header record that contains the following elements:
 - ◆ **Document ID** – The unique ID that identifies each individual document.
 - ◆ **Document Type** – States the type of document such as an invoice, order, confirmation, etc.
 - ◆ **Version** – The version number of the document type.
 - ◆ **Trading Partner ID** – Identifies the trading partner owner of the document.
 - ◆ **Mailbox** – Contains the owner mailbox name.
 - ◆ **User** – If a user that owns a document, this will contain their User ID.
 - ◆ **Origination Date/Time** – Indicates what date and time the document was received at or created by the VNI server.
- Target (02)** – A document may contain from zero to any number of Target records. These identify to whom a document will be sent. Each target will identify a single document destination.
 - ◆ **Source** – Indicates the source of the target. May contain the word "Auto" indicating that the target is a result of the Auto-Target process. Can also contain a User ID indicating who added the target.
 - ◆ **Trading Partner ID** – Identifies a destination trading partner.
 - ◆ **Mailbox** – Contains the ID of a destination mailbox.

- ◆ **Trading Partner Name** – Contains the name of the targeted trading partner.
- ◆ **Send Date/Time** – Indicates the date/time when a document is actually forwarded to the target. This element remains blank until the document is sent.
- **Comment (03)** – A comment record is always created directly by a user. These are for viewing purposes only and don't constitute usable data.
- ◆ **Send** – Indicates whether the comments should be included when they are forwarded to their target destinations.
- ◆ **Text** – Contains the text of the user comment.
- **Message (04)** – Messages are placed in a document by the system when some event occurs where a user needs notification of some event.
- ◆ **State** – Indicates whether or not a message is critical. (Documents may not be sent so long as there are critical messages.)
- ◆ **ID** – May contain a message ID for standard messages.
- ◆ **Text** – Contains the text of the message.

Document Definition Files

The structure of each VNI form document will be defined in a tabular form and stored in Document Definition Files (DDF). Each will dictate the structure of a single version of a VNI form document. This structure will allow VNI to evolve document structures and still provide usability of older versions of documents. Each time VNI alters the form of a document, a new DDF must be produced so that the system utilities can decipher its contents.

A case in point, a trading partner may have many order documents archived at any given instant. Older order documents may be of a different vintage than newer ones. Because VNI document processing utilities, such as translation programs and screen forms, interact with documents indirectly through a mapping class that decipher the documents per the instructions of the DDF, the user may be completely ambivalent about the operations they may perform against which version of document. Even the VNI document utilities are blind to minor changes in document structure since they don't directly interact with the document files themselves, rather with objects that provide a level of indirection between the physical document files and themselves.

Each DDF will contain the following information regarding a specific version of a document type:

- The types of records contained in a document.
- The ID of each record type.
- The nature of each record type (Optional, Required)
- The relative hierarchy (looping structure) of the record types.
- The name of each data element in a record.
- The position of the data elements within the record.
- The type of each data element (Numeric, Alpha, Alphanumeric, Date, Money, etc.).
- The maximum and minimum size of each data element.
- The nature of each data element (Mandatory, Optional, Conditional)

By convention, DDFs will reside in a special directory and will be named according to the document type and versions they represent. Their filenames will be structured as follows "TTTTTTT.VVV.def" where 'T' represents the characters of the document type and 'V' the digits in the version of the document type. Thus a typical document definition filename might be "order.001.ddf".

Document Key Files

As stated in previous paragraphs, it is crucial to our throughput and accessibility strategies to keep our documents in ASCII flat files. However, this poses the problem as to how we can support the need for ad hoc queries against documents by clients and the easy collection of their key values for client presentation.

It is our contention that on any given document type, there will not be very many data elements which would be considered as a distinguishing characteristics, or “keys”, and be used conditionally within an ad hoc query. Let’s take an affidavit for example, some distinguishing data (key) elements might be the invoice number, advertiser name, invoice date, agency name, station, etc. But almost no one would be interested in conditionally querying against or viewing in summary the 1st agency address line, 2nd address line, agency-commission, network, etc.

In this system we will support both lightweight document storage and heavyweight ad hoc queries by identifying the key data elements for each document type and storing instances of those key elements into a database table.* Taking this approach, we don’t have to create a special table for each document type in the database, we can store they key element instances with a purely table driven strategy.

The strategy involves defining which data elements are “Key” to each document type in a Document Keys File (DKF). The key data elements are listed by name, one per line, in the DKF. The names must correspond to the data element names as defined by the DDF (See *Document Definition Files* above.). Since the data elements considered “Key” for a document type will transcend versions of document types; and since database storage of key elements has moderate implications **, the list of key elements for each document type will be stored in separate tables (files) from the document definitions (DDFs).

With these instances of key elements defined in a DKF and the actual values stored in a database table, it is very simple operation to perform ad hoc queries against them. We also avoid weighing down the system with complex, large, cumbersome, and curiously intertwined database tables that, aside from providing document storage, represent complex data relationships embodied within each document. Thus, when a request to build a customer view of orders for “Johnny’s Agency” is forwarded by a client, the server will be able to quickly respond by querying the DBMS for key element instances stored in a simple lightweight table. When a client requests a summarized view of the documents contained in a folder, the server can easily determine which are the key data elements for the document type and extract only those from the documents and return them to the client. When VNI operations needs to add support for a new document type, it becomes a simple matter of creating a DDF and a DKF for the new document type. A simpler process than generating entity relationships and designing and normalizing database tables just to store documents and to identify a few key elements.

We should note that only data elements in records of the 1st order (one instance per document) may be defined as keys. Data elements resident in records where there may be multiple instances of that record type in a document, and therefore multiple instances of a key within a document, can not constitute a distinguishing characteristic of a document. For example, a ‘Network’ data element in an affidavit resides in the ‘Schedule Line’ record. There can be many instances of ‘Schedule Line’ records in an affidavit. Therefore displaying the ‘Network’ of a document is not only ambiguous (because there can be many of them) but also not a distinguishing feature of any affidavit.

*In the initial release of EC@VNI, before ad-hoc query capability is added, the document keys won’t be stored in a database table.

**If needed, we can easily rebuild the database keys table from the appropriate DKF and document files.

Error! Bookmark not defined. Server Directory Structure

- “/ec” - All storage for **Error! Bookmark not defined.** Server will be organized under this tree, also referred to as EC_ROOT.
- “EC_ROOT/bin” – Will contain all binaries.
- “EC_ROOT/defs” – To store all document definition and key files.
- “EC_ROOT/script” – Holds all scripts invoked directly by the Document Manager.
- “EC_ROOT/log” – Storage for transaction logs

- “EC_ROOT/<tp_code>” - A trading partner’s base directory. A single directory, named by TP-Code will be created for each trading partner. Each trading partner (TP) directory will contain sub-directories for each mailbox; ie “in”, “out”, “sent”, “arch_in”, etc.
 - “EC_ROOT/<tp_code>/<mailbox>” – Mailbox directories that will hold document files.
 - “EC_ROOT/<tp_code>/discard” – Will hold discarded (deactivated) document files until they are purged. When an operator deactivates a document, the server will move it to this directory and flag it “Inactive” in the database. When the server does routine purges, the documents will be deleted and pertinent database entries purged.
 - “EC_ROOT/<tp_code>/work” – A working directory that will hold documents where an operation is pending. For example, a translator will copy a document file here just prior to asking **Error!**
- Bookmark not defined.** to register it. When the server registers it, it will copy it from the work directory to the appropriate mailbox directory while giving it a new file name and attaching the appropriate document header record.

All active documents on the server will be stored in a mailbox sub-directory. **Error! Bookmark not defined.** only will operate on documents that are appropriately stored in a mailbox directory.

Standard Mailboxes

Each trading partner will be configured with several standard mailboxes. Each has a specific purpose and is not optional, configurable, or capable of being renamed, as far as the client is concerned (at least initially). Documents may be moved between mailboxes as various operations are performed against them.

As previously stipulated, each mailbox will consume a sub-directory underneath the trading partner subdirectory. So a typical path for an “In” mailbox might be “EC_ROOT/acme_cable/in”. Underneath each mailbox directory the document files reside.

Mailbox	Direction*	Description
Up	Inbound	Inbound application document batches will be stored here.
Down	Outbound	Outbound application format documents will be stored here for downloading.
In	Inbound	Newly received documents will reside here until they are downloaded or deleted.
Out	Outbound	Newly uploaded documents will reside here until they are sent to their specified target(s).
Rcvd	Inbound	Inbound documents will reside here once downloaded by a user.
Sent	Outbound	Outbound documents will be move here after being sent.
Arch_in	Inbound	Archived inbound documents will reside in this mailbox until purged.
Arch_out	Outbound	Archived outbound documents will reside in this mailbox until purged.

*Direction is relative to the client.

Document Filenames

Document filenames will consist of two or three parts delimited with periods (.):

- Document ID** – The assigned Document ID
- Document Type** (abbreviated) – an abbreviated form of the document type, for example, “inv” for invoices, “ord” for orders, etc.

- Account ID** (optional) – In cases where a document is targeted to a specific account within a trading partner, that Account ID instance will constitute the 3rd and final segment of the document's filename.

A fully qualified path for a document file, targeted to a "billing" account, might look like: "/ec/acme_cable/in/107701.inv.billing". Whereas a document that hasn't been assigned to an account might be: "/ec/jdoe_agency/in/102394.ord".

Concurrent Document Access

Given the simultaneous multi-user nature of our application, it is imperative that we allow simultaneous access to documents by multiple users. But we must also protect the integrity of the documents themselves by disallowing potentially conflicting operations to be taken against them. For example, two users may nearly simultaneously retrieve the same document. Let's say the 1st user makes some changes to the document and then stores back to disk at the server. The 2nd user may wish to send the document after viewing but some changes have taken place since he reviewed the document. This has the potential to perform an unwanted operation.

We resolve the issue through the use of time stamps. When any operation is invoked against a document by the server, whether it affects the document store or not, it touches the document file so that it is time stamped. Whenever a client retrieves a document, the server attaches an ID record at the start of each document that contains its ID, its location, and the time stamp on the document file when it was read. If any operation against the document is subsequently invoked, the client passes back the ID record, which contains the time stamp of its latest read. The server will then compare the time stamp against the document file's current time stamp. If it has changed, the server rejects the operation back to the client informing them that changes have taken place since their last read.

Additionally, whenever an operation is in-progress on a document, the server puts an exclusive lock on the document file so that it can't have any other operations invoked against it. As soon as the operation finishes, the server releases the lock.

This approach doesn't require us to lock any files for any extended periods of time (only while a transaction is in-progress). It also permits concurrent access by an unlimited number of users. No database access is required whatsoever, and touching a file is very cheap in terms of system resources. Users can retrieve a document for editing purposes and leave it open on their workstation for hours without hurting the system whatsoever. No complex scheme is required either to notify the users of updates. The users aren't inconvenienced, except in very rare circumstances when a document is changed underneath them. In the very worst case they may have to re-read the document and re-invoke their operation.

View Files

Whenever a user defines a document view, a "<view name>.view" file is created that stores the crucial elements of the query, that in essence, define the view. A second file is created, whenever the query is executed, name "<view>.view" which contains the document ID's of each document matching the query criterion. So if, for example, a user defined a view named "affidavit", the document manager would create a file named "affidavit.query". When the user populated the view, a second file would be created named "affidavit.docs". Both of these files would reside in the user's home directory (EC_ROOT/<tp_code>/<user name>).

Database Tables

All tables will reside under the "ec" database volume.

Document Operations Manager

When we speak of **Error! Bookmark not defined.** server, in many cases we refer to the Document Operations Manager (DOM). It lies at the heart of the server's functionality by managing all of the communications and connections to client instances and by managing and invoking operations for them request. It also centralizes access to the database.

Transactional Integrity

It is necessary for DOM to insure the integrity of each transaction executed. Since all transactions are script driven and many, if not most, occur mostly outside the context of a DBMS, DOM must insure the relational integrity of the system in the event of a mid-execution failure. Given that each step of each transaction is logged in a transaction log file (See the ExecOperation request below.), DOM can tell which transactions were not properly terminated.

Upon startup, DOM checks the "EC_ROOT\log" directory for any transaction log files. If any are found, it calls up the appropriate operation from the Operations table, and invokes the specified Rollback Script, if there is one. These Rollback Scripts are responsible for cleanly backing out transactions. When the script successfully returns, DOM will delete the transaction log file itself.

DOM will not respond to login requests, or any other messages, until it has attempted to reverse all partial transactions.

Client Requests Overview

A DOM will receive and operate on several standard requests from an **Error! Bookmark not defined.** client. DOM will always return the value SUCCESS (and potentially a result set) when it succeeds. It will return FAILURE and a text error message when it fails.

All of the operations are executed natively by DOM (or one of its threads) except the ExecOperation request. These are executed through shell scripts that, by convention, return success or failure values to DOM.

- ❑ **Login** – Establishes a connection to DOM.
- ❑ **Logout** – Destroys a connection to DOM.
- ❑ **GetDoc** – Tells DOM to retrieve the specified document(s) by its ID.
- ❑ **GetShortDoc** –Asks DOM to retrieve an abbreviate version (key data elements only) of the specified document(s).
- ❑ **StoreDoc** –Requests that DOM re-write the passed document.
- ❑ **AddDoc** – Asks DOM store a new document.
- ❑ **DeleteDoc** – Tells DOM to delete the specified document.
- ❑ **Move Doc** – A request for DOM to move a document to another mailbox or state directory.
- ❑ **GetKeyNames** – Requests that DOM return the key element names and data types associated with the passed document type. This information will serve as an aid for helping users to create custom document views.
- ❑ **BuildView** –Asks DOM to build a document view, under the specified name, and return the key elements of the resultant documents.
- ❑ **GetView** – Instructs DOM to return the key elements of the documents associated with a particular view.
- ❑ **GetQuery** – Allows the client to retrieve the contents of a query that formulate a document view.
- ❑ **StoreQuery** – Asks the server to store the contents of a document view query under the specified name.

- ❑ **RegisterDoc** – A request for DOM to register a Document (assign it a Document ID) that is stored under the specified name in the user's upload directory. This is normally invoked on transmission batches that were previously transferred via FTP to the upload directory.
- ❑ **GetTargets** – Asks DOM to return a list of authorized targets for the specified document type, so the client can have the user select them from a list.
- ❑ **AddTarget** – Makes DOM add a target to a document.
- ❑ **DeleteTarget** – Tells DOM to delete a specified target from a document.
- ❑ **GetOpsList** – Requests DOM to return a list of the operations supported for a specific document type.
- ❑ **GetFolders** – Returns a list of folders available to the specified user.
- ❑ **ExecOperation** – Instructs DOM to execute a stored operation on the document(s) specified by the passed Document ID(s). Examples of stored operations would be commands like "Send", "Validate", etc. DOM depends on scripts, registered as valid operations in the Operations table in order to carry out the requested operation.

Message Format

Request and return messages use the same format and are variable in length, with 16 bytes required, 12 in the header and 4 in the tail. All messages will use the following format:

- ❑ **Client ID** – 4 bytes, offsets 0 - 3
- ❑ **Request or Return Value** – 4 bytes, offsets 4 - 7
- ❑ **Bytes in Buffer** – 4 bytes, offsets 8 - 11
- ❑ **Buffer** – Length specified by Bytes In Buffer (offsets 12 - n)
- ❑ **End of Transmission (EOT)** – 4 Bytes (immediately following buffer)

Both client and server will retrieve packets in two steps. In the first step, they retrieve the first 12 bytes of the message, determining if the request or return value is valid and getting the length of the buffer. Once they know it's a valid message, they will go ahead and retrieve the buffer and EOT. They store the buffer contents in allocated memory, parse it into its components, and process it.

In request messages (server bound) the 2nd element holds the request value, while in result messages (client bound) it contains a request return value. The return value can be zero, or SUCCESS, a positive integer indicating a critical error, or a negative integer indicating a non-critical error. On any error return, critical or not, the buffer always contains a text error message describing the condition.

In any foreseen cases, the Buffer would always contain ASCII data exclusively. When numerous data elements are included as part of a request packet (server bound), they will be pipe ('|') delimited. In result packets (client bound) data elements are also pipe delimited. But often result packets will store result sets in buffers that will include not only data elements, but whole records and documents as well. In these cases, the buffer will have newline ('\n') delimited records, and form feed ('\f') delimited documents. EOT will always signal the end of the buffer, as well as the entire message.

Script-based Operations

As alluded previously, most client requests are executed internally and directly by DOM or one of its threads. This works very well for limited and very concise operations, such as storing and retrieving documents, building document views, etc. But there are so many operations that may need to be performed against a document that it is difficult to create a special request message for each. Additionally, there are some operations such as translations where, at least partially, we are using 3rd party products to perform them (The product we have operates primarily from a command line). Nor can we anticipate every operation that may have to be performed against each document type for even a single trading partner. When you factor in the number of trading partners we're going to serve, the potential operational diversity is enormous. We expect to be writing many custom programs to satisfy various large trading partners' unique requirements. Finally, we will have a number of different people working to create these custom

solutions and we wish to give them as wide a latitude as possible within the confines of auditability, transactional integrity, and good form.

The best way for us to satisfy all of these requirements is to utilize the scripting languages native to the UNIX operating system. But we must tie these custom scripts together into a cohesive framework that provides support for the diverse operations. Thus DOM supports a special client request called "ExecOperation". The client tells DOM which script to execute by specifying an operation code. All major operations that translate and route documents will be executed through one of these shell scripts.

The scripts are executed by a DOM service thread, but not natively with compiled C++. It actually invokes an instance of a shell and instructs it to execute the script. The DOM thread will start a transaction in the database and create a log file that will help it to determine whether a transaction, executed via a script, has successfully completed. We explain in the Transactional Integrity section that each script program has a reversing counterpart (rollback) script that can undo a transaction that is partially completed by its associate script. No script may be registered as a valid operation without a corresponding rollback script.

A script is registered in the system when it is called out in an operation record (Stored under the Operations table by the DBMS). Operation records are unique to each document type. In other words, you may have a "Send" operation for 20 different document types, but there may be a different script associated with each. There is no limit to the number of operations that can be assigned to a document type. Each operation names a single execution and rollback script. The execution script is called when the client issues an ExecOperation request naming the operation. The rollback script is called exclusively DOM on startup, if there is an incomplete transaction lurking in the system. Each script is responsible for updating the transaction log file so that its rollback counterpart can do its job properly.

The client is responsible for passing the appropriate script arguments to the server so that it can in turn pass them along to the script. There is no way feasible for DOM to be aware of the number and types of arguments required by a script, so it will be the client's responsibility to supply them.

Request Messages

Request messages are sent by **Error! Bookmark not defined.** client instances to the Document Operations Manager requesting some action be taken. The first message sent by a client must be a login request that establishes a connecting socket through which the client and server (DOM) will subsequently communicate. The client is responsible for terminating the connection when has completed operations using the Logout request.

On all requests except the Login and Logout requests, a DOM thread that is manning the client connection, will always check to see if the current user is authorized to perform the requested task. It will reject requests a user is not authorized to perform.

In all cases following a request, DOM will respond with a result message indicating success or failure. Failure comes in two flavors, critical and non-critical. When a non-critical error is returned, the client should check the returned result set.

Login Request

Before an **Error! Bookmark not defined.** client instance can perform any operation or view any document, it must first connect to DOM. It does so through a TCP socket at DOM's assigned primary port following these steps:

- ❑ DOM receives a login request from a client that is attempting to connect passes its client ID, user name, and password.
- ❑ DOM assigns an available port to the specified client ID and registers them both in the port registry.
- ❑ DOM spins a new thread assigned to a secondary port passing it the client ID, user name, and password.
- ❑ The new thread opens the secondary port and waits on client requests.
- ❑ After spinning the thread, DOM returns to monitoring the primary thread for login requests.
- ❑ The thread verifies the user name and password against entries in the "User" table.
- ❑ If the user name and password are verified, it will confirm a connection to the client by returning the SUCCESS and newly assigned port number, otherwise it will return FAILURE and the thread will follow the Logout procedure (See Logout Request).
- ❑ It stores the user name and password to use them for subsequent security validation.
- ❑ The connection thread will stay active monitoring the port for incoming requests until:
 - ◆ The user logs off
 - ◆ The connection has been inactive for more than the time-out period
 - ◆ DOM terminates the connection because the same client ID requests a new login.

Logout Request

As soon as this request is received by a DOM thread:

- ❑ The thread returns a SUCCESS result to the client with no accompanying data.
- ❑ It blocks on exclusive access to the port registry.
- ❑ It de-registers itself and the client ID.
- ❑ It closes the port.
- ❑ Lastly, the thread exits, terminating execution.

GetDoc Request

This request is sent by the client to retrieve selected documents in their integral form. Each document requested is by its document ID that is copied, pipe-delimited, into the buffer portion of the message.

Once the message is received and validated and the user is authorized by a DOM thread it:

- ❑ Begins a loop operating on each document ID.
- ❑ Queries the Document table to determine the storage location for each document.
- ❑ Reads the document file time stamp
- ❑ Builds a header record for it containing the document ID and the time stamp and stores it in the allocated memory buffer.
- ❑ Reads document from file and stores properly delimited in the memory buffer.
- ❑ Loop ends after the last document ID it received
- ❑ Builds an appropriate result message with the documents contained in the buffer area
- ❑ Sends the result message.

The thread will return SUCCESS value if it succeeds in loading all requested documents. If a single document has been requested and it fails to load the document, it returns a critical error. Finally, if multiple documents are requested and at least one document is successfully loaded, it returns a non-critical error.

GetShortDoc Request

This message is used by the client when it needs to present an abbreviated view of one or more documents and it knows their document ID's. The server will return a result set containing only the key element instances of the specified documents, i.e. invoice numbers, advertiser name, agency name, etc. The client will populate the request buffer with pipe-delimited document ID's for all of the desired documents.

After the server receives and validates the request message and checks the user's authorization, it formulates a SQL Query that includes all key element instances for the desired documents. After invoking the query it retrieves the elements and copies them with their document ID into a memory buffer. In following conventions, it will delimit the key element instances from the document ID's with pipe symbols ('|'), key rows with newlines ('\n'), and documents with form feeds ('\f').

After populating the buffer, it builds a result message and returns it to the calling client. The message will return a SUCCESS value only if it's able to load the keys from all requested documents. It will return a non-critical error on partial loads, and a critical error when it fails to load keys from any of the requested documents.

StoreDoc Request

A client will make this request when it wishes to save one or more preexisting documents after changes have been made. The client will build a request message containing the whole, delimited contents of each document it wishes to store. It must include public and private comment records because any previously existing comment records will be overwritten. Each document must start with a valid ID record containing the server assigned document ID, file location, and the timestamp of the client's last read. Once it has appropriately built the request message, it will forward it to a server thread.

When the server thread receives and validates the request and checks the user's authorization, it will perform the following on each document:

- ❑ Compares the timestamp the client passed, in the ID record, to the current time stamp on the document file. If it has changed, it returns a critical error message informing the client that the file has changed since their last read.
- ❑ It then attempts to issue an exclusive lock on the document file. If it fails, it returns a critical error to the client because an operation, against the document, is in-progress.
- ❑ An attempt is made to write the file to its appropriate location. If that fails, it returns a critical error.
- ❑ It calls the key extractor that will locate its key elements and update the KeyInstances table in the database.

Once the server successfully completes the previous steps for each document, it builds and sends a return message indicating success.

AddDoc Request

A message a client issues when it wishes to store a single brand new document in the system. The client will prepare by building an ID record containing the target mailbox and state directory. The remaining elements of the ID record should be blank. The server will populate them after it stores the document.

When the DOM thread receives the request, it takes these actions:

- It inserts a new document record in the Documents table. The DBMS will create a unique ID for the document. Failure to insert the document record in the table constitutes a critical error.
- It saves the passed document in a file with the same name as the new ID and in the mailbox/state directory location specified by the client.
- It calls the key extractor to update the KeyInstances table with key elements from the new document.
- It builds a return message that includes the ID record, now populated with the document ID and timestamp of the document file.
- The return message is sent back to the client indicating success (assuming everything went according to plan).

DeleteDoc Request

In this message the client must pass an ID record containing the document ID and the timestamp of when the client last read it.

The DOM thread that operates on the delete request will only do so if the user is authorized to delete documents from the mailbox. Once it receives the request, it attempts to place an exclusive lock on the file. If it fails it returns a critical error indicating that an operation against the document is in-progress.

Once it successfully locks the file, it deactivates its pertinent records in the database. When that has successfully completed, it moves the document file to the deleted directory for the mailbox and returns a message to the client.

GetKeyNames Request

The client uses this message to retrieve the names and data types of the key elements for the specified document type. When the client needs to set up columns in a viewer or when it needs to formulate criterion for a document view, it can use this call to retrieve key column names for a document type.

When a DOM thread sees and authorizes the request, it queries the DocTypeKeys table for the names and data types of the key elements for the selected doc type. It then pipe delimited the elements, stuffs them in the buffer of the return message. Unless DOM is unable to query the database, it will return SUCCESS.

BuildView Request

The client uses this message to build custom document views for a user. The message buffer must contain the document type and the key criterion used to build the view. For example, "AdvertiserID = 10035 AND InvoiceDate >= '10/31/97'". The criterion section of the message buffer must form a valid "WHERE" clause of a SQL expression. The server will complete the other portions of the SQL expression such as the "SELECT" and "FROM" clauses.

After receiving and validating the message, the server thread does the following:

- It builds a SQL statement filling in the portions the client didn't in order to get a result set back from the DBMS that contains all of the document ID's that meet the client's criterion.
- It then issues the SQL statement to the database and retrieves the result set.
- It creates the view files with name "<view name>.query", that will contain the query text, and "<view name>.docs", which will hold the list of document ID's.
- It then walks through the list of document ID's, identified by the SQL query, and copies their key elements, appropriately delimited, into a message buffer.
- It then returns the message buffer with all of the records in the client.

The reason that we keep view files is so that a client can later just look at the view of the documents without having to re-query the database. The client can also retrieve the query text and view it should they wish.

GetView Request

A client will send this request when either a view has previously been created, in which case it will supply a view name, or it wishes to retrieve key elements for all documents and a given state, in which case it supplies the mailbox and document type. In either case, the server will return a result set that includes key element instances for all of the documents named by a view or state/doc type.

After receiving the request, it will determine if it is dealing with a view. If so, it will open the "<view name>.docs" file and retrieve the key element instances for all documents contained therein. If the client has requested for a state/doc type combination, it will look at all of the documents sitting in that location and retrieve the key instances for them.

After retrieving the key instance information, it will pack them appropriately delimited into a message buffer returning the result to the client.

GetQuery Request

This message allows the client to retrieve the query text associated with a particular view. The client will put the view name in the buffer portion of the message.

When the server receives the request, it simply retrieves the contents of the file "<view name>.query" and place it in the buffer portion of a return message.

StoreQuery Request

Provides the means for a client to update the contents of SQL query related to a particular view. In this case the client places the view name and the entire next of the updated query in the buffer section of a request message.

Upon receiving the request, the server will attempt to gain an exclusive lock on the file "<view name>.query" and rewrite it with the updated contents.

RegisterDoc Request

Typically a client will upload transmission batches, they wish to process, into the upload directory belonging to their user. DOM will not process a transmission batch until it has been registered in the system as a document. This facility will allow the client to register a batch, once it has been uploaded. The client must initialize the buffer portion of the request with an ID record containing the filename, document type, mailbox, and the state directory in which it should be stored.

After receiving the request, the DOM thread will try to find the file and move it to the appropriate directory. It will then add a row to the Document table, which will create a unique document ID. Then it will build a return message where the buffer contains the ID record completed with the new document ID. After the client receives the message, it will then be free to invoke qualifying operations against it.

GetTargetList Request

A request a client makes when they are trying to present a user with a list of possible target trading partners for a document type. In order to send this request, the client must pack the buffer section. It must contain the document type being addressed, a qualifying trading partner type, and an indication, as to whether it wants all potential trading partners, or just the ones it has targeted previously.

When the server receives the message, it will either query the TPXRefs or the MBDocTypes table for potential trading partners and mailboxes. It will query the TPXRefs table if the client only wants to look at trading partners it has used in the past. Otherwise it will query the MBDocTypes table to get a list of every potential trading partner and mailbox for the specified doc type. When the results are returned from the database, it packs them in message buffer and sends them back to the client appropriately delimited.

GetTargets Request

A client will make this request when it needs a list of the targets assigned to a specific document. It must pack the document ID into the buffer portion of the request message.

The server will query the DocTargets table and respond with a complete list of all the targets selected for the document including the tp_code and mailbox for each.

AddTarget Request

With this request, the client instructs DOM to add a target to the specified document. The client must pack the document ID, target trading partner, and target mailbox into the buffer portion of the request.

After receiving the request, DOM will attempt to insert the desired target into the DocTargets table.

DeleteTarget Request

Using this message, a client may request that the server delete the target, specified in the buffer section, by its tp_code and mailbox.

Once DOM receives and parses the message, it will attempt to delete the selected target from the DocTarget table.

GetOpsList Request

Allows the client to download a list of the script-based operations supported for a particular document type. The client will use this list to know which operations are legal for a specific type.

DOM will respond to this request by querying the Operations table and returning all of the valid operations for the document type specified by the client.

GetFolders Request

Returns a list of the folders (state directories/doc types and user defined views) available to the specified user. This is so that the client can display the folders available to each user.

ExecOperation Request

See Script Based Operations section for a full description of this request

The client will pack the buffer portion of the message with the document ID's and desired operation to be invoked. The client must also pack any arguments required by the execution script.

When the server receives the message, it will unpack the document ID's and invoke the script indicated by the operation code for each.